

Vehicle Make and Model recognition

Lecturer: Eran Edirisinghe

May 2021

COP507 – Computer Vision

By Leon Davies

Student ID: F034961

Table of Contents

[Explanation of Block Diagram 4](#_Toc73044791)

[Task A: Block diagram of VMMR system 4](#_Toc73044792)

[Design of the VMMR system 5](#_Toc73044793)

[*System setup* 5](#_Toc73044794)

[*Predictions on real world data* 7](#_Toc73044795)

[*Operation of the VMMR* 7](#_Toc73044796)

[Assumptions 7](#_Toc73044797)

[Task B – Implementation of VMMR system 8](#_Toc73044798)

[Task C – Test VMMR system of car dataset 9](#_Toc73044799)

[Discussion of testing, implementation and results 9](#_Toc73044800)

[Experiments 13](#_Toc73044801)

[*Bag of features model* 13](#_Toc73044802)

[Model with little data cleaning 13](#_Toc73044803)

[*Testing the model on Real World data* 15](#_Toc73044804)

[Analysis of best model accuracy 18](#_Toc73044805)

[19](#_Toc73044806)

[Precision and recall of data cleaned model 20](#_Toc73044807)

[Conclusions of the VMMR 22](#_Toc73044808)

[Task D – Convolutional solution for a VMMR 24](#_Toc73044809)

[Task E – Increasing data diversity. 26](#_Toc73044810)

[Appendix 27](#_Toc73044811)

[Appendix A: Code and Output for VMMR system implementation 27](#_Toc73044812)

[Code for VMMR System 27](#_Toc73044813)

[*Vehicle Make and Model Recognition 27*](#_Toc73044814)

[*Setup of file 27*](#_Toc73044815)

[*Cleaning of the Dataset 28*](#_Toc73044816)

[*Test if pre-processed elimination was successful 29*](#_Toc73044817)

[*Splitting of data 30*](#_Toc73044818)

[*Data processing 32*](#_Toc73044819)

[*Samples of training/testing data 33*](#_Toc73044820)

[*HOG feature extraction 35*](#_Toc73044821)

[*Fitting model 36*](#_Toc73044822)

[*Visualisation of Testing data results 37*](#_Toc73044823)

[*Evaluation of data on Real-world data 39*](#_Toc73044824)

[*Visualisation of Real world testing results 40*](#_Toc73044825)

[*Function to show 10 random samples of data 41*](#_Toc73044826)

[*Pre-processing function 42*](#_Toc73044827)

[*Function that builds Recall and Precision confusion matrices 43*](#_Toc73044828)

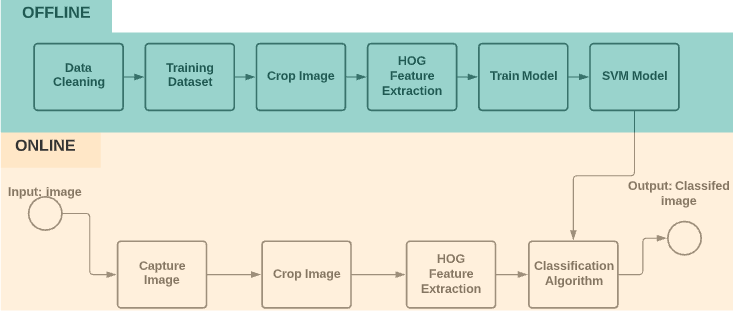
# Explanation of Block Diagram

Task A: Block diagram of VMMR system

**Figure 1** shows a block diagram of the VMMR (vehicle make and model recognition) system, it includes two streams of blocks. The first stream of blocks is currently in an ‘offline’ state – this means it is currently inactive. This stream shows the creation of the SVM model. The first step of this stream is to clean the dataset, an analysis of the images within each class will be done, images of low quality and duplicates will be eliminated through computer vision implementation. The next block is the load in of the training dataset, in this scenario the dataset holds images containing cars sorted out into 27 different make and model combinations, each make-model combination contains 10 images, the labels from these cars are determined by the folder name that they reside in. In the next block these images are then cropped to only contain the front view of the vehicle including the bumper, headlights, grill, and badge. These cropped images are then parsed to the next block where HOG feature extraction occurs. The feature extraction will result in a 2D array of HOG features that will be used to train the model along with the labels for each 2D array. The SVM model will produce a classification algorithm that can be used to classify images of cars that match the make and model of the training dataset.

The second stream of blocks is in an ‘online’ state, the first block is Capture Image, it is given an input of an image of a front facing car, in the next block this image is then cropped to only contain the features at the front of the car (as mentioned above). This cropped image is then sent to the next block where HOG feature extraction occurs, the HOG features represented in a 2D array will then be parsed to the next block that contains the classification algorithm. The classification algorithm will then output the predicted make and model of the car.

**Figure 1**: Block diagram of the VMMR system



# Design of the VMMR system

The VMMR system has been briefly described in the explanation of the block diagram, this section will elaborate on the overall design decisions and choices of algorithms and settings.

## System setup

The dataset used for this project has been provided by the stakeholder, it contains a total of 2117 images of 27 different car makes and models. The images contain slightly different viewpoints of cars, augmented data with skewed brightness and contrast, images with varying degrees of noise, images with borders, and duplicate images. The images within each class of data will be analysed for quality control, any images that cannot be improved for classification through pre-processing will be eliminated, this is important to be done as training the model will require as high quality images as possible, duplicate images can result in an artificial accuracy as the test portion of the dataset may contain duplicate image that reside in the training portion. The dataset will be loaded in and stored in MATLAB in an [imageDataStore](https://www.mathworks.com/help/matlab/ref/matlab.io.datastore.imagedatastore.html) object, this will be done as it enables the usage of processing all the data at once which decreases computational cost and increases efficiency, it also keeps the data in an organised format where labels can be directly imported from folder names, certain data types (such as PNG) can be ignored, and data can be shuffled simplistically. The entirety of the dataset will been split into training and testing data in the split of 70/30 (training, testing).

The *imageDataStore* object containing the dataset is then parsed to a cropping function where all images are cropped to only contain the key features at the front of the car, this was done as these features contain enough information and are representative of the car make and model, furthermore the stakeholder requested that only this region of the car is required for the classifier model.

HOG (histogram of oriented gradients) feature extraction is then done on the dataset. HOG is a feature descriptor that can identify the structure or shape of an object, it does this through extracting the gradient and orient of the edges in an image, this information is then used to create a histogram. This feature extraction algorithm was chosen for this task as the images of cars contain slightly different shapes of certain features such as grills and headlights, HOG can identify these differences well by identifying the gradient and orient of the edges for each feature. The front view of different cars contains mostly the same features (headlights, grill, etc) however different make and models contain slightly different shapes and orients of these features which indicate that HOG would prove beneficial in this scenario. HOG can be implemented in MATLAB with the integrated function [*extractHOGFeatures*](https://www.mathworks.com/help/vision/ref/extracthogfeatures.html)  which requires input of an image matrix, it will extract the HOG features from it an output a 2D array. The output of the HOG extractor can be used to train the classification model. The cell size of the HOG extractor refers to the size of each HOG cell which also corresponds to the amount of feature cells – for large images with a higher amount of spatial information a higher cell size is required, however for this task as features are small, a smaller cell size will be better. For this task, a cell size of [6 6] has been chosen through experimentations, as seen in **Figure 2.** It was found that a lower cell size than this would not increase accuracy but would greatly reduce computational efficiency.



**Figure 2**: HOG Features plotted on top of corresponding image

The model architype being used for this classification task is an ECOC (error-correcting output codes) classifier using the binary learner SVM (support vector machine) model. A SVM is a supervised machine learning model that can be used for non-probabilistic linear binary classification, it computes binary classification through the usage of a decision boundary or hyperplane in a 2D plot, each side of the plane represents a class, the object of training the model is to provide the most optimum hyperplane which separates data values between separate classes. A SVM model was used for this task as SVM’s work well with unstructured data – like images, SVM’s can compute complex relationships between different datapoints which is beneficial when using HOG data of the car images as many of the features, such as the shape of the bonnet, will be similar and the most different features will be very specific and small, such as the shape of the badge or grill – SVM can calculate the relationship between the shapes of these specific features within a class while identifying the similar characteristics in the opposing classes, placing the hyperplane in a location that clearly identifies the separation between the different arrays.

ECOC is a classification model that can utilise a binary classifier (such as an SVM) to compute multi-class classification. It does this through enabling a one-verses-one computation, enabling each classification of an image to be computed as several binary classifications, the additional binary models are then used for error-correction. ECOC can be implemented in MATLAB using the built-in function [*fitcecoc*](https://www.mathworks.com/help/stats/classificationecoc.html#bug0_3g-1).

To further improve the quality of the model cross validation will be used. Cross validation is a technique within machine learning that enables resampling of test and training data, it does this by shuffling the dataset and partitioning the test and training data, it will then fit the model with the partition of training data and evaluate it on the test data partition. For this project 10 fold cross validation will be implemented, this means that the procedure of cross validation will be computed 10 times. This number of folds was chosen as the dataset will not be large enough after data-cleaning to justify computing any more folds. Cross validation will be implemented in MATLAB using the built in function [*crossval*](https://www.mathworks.com/help/stats/crossval.html)*.*

As the dataset will be stored in an *imageDataStore* object pre-processing will be done to the entirety of it at once. Once the test data is partitioned and pre-processed it will be converted into a 2D array of HOG features, this will then be used with the model built to make predictions, predictions will be made using MATLAB’s built in function [*predict*](https://www.mathworks.com/help/stats/compactclassificationdiscriminant.predict.html), the HOG features will be computed with the algorithm and the model will provide a predicted label of what it classified it as, this can then be computing alongside the ground-truth labels of the test data to calculate the accuracy, as well as the precision and recall, results will also be plotted in a confusion matrix.

## Predictions on real world data

To evaluate the overall quality of the model real-world testing data will be produced through finding images of cars that are the same model and make that the model was trained to classify. These image will be stored in an *imageDataStore* object and parsed through the procedure shown on the block diagram.

Once parsed into the *imageDataStore* the images will be cropped to include only the front end of the vehicle containing the key features that the model was trained on. The images will then be converted into 2D arrays and undergo HOG feature extraction – this data will then be parsed into a function that predicts the label of the image and calculates the accuracy, precision, and recall. Through using real-world data, the model can be more accurately evaluated, as this data will be slightly different to the training and testing data, issues such as overfitting or artificial accuracies can be identified which can help produce an overall higher quality VMMR model.

## Operation of the VMMR

The VMMR will capture an image of the front view of a car which matches one of the 27 make-model combinations that it was trained on, it will then crop this image and apply pre-processing to it to normalise to a specific state. It will then apply HOG feature extraction to the image, transforming it into a 2D array of HOG features. These HOG features will then be parsed into a prediction function where the trained model will analyse the HOG features and make a prediction of what car and model it is.

## Assumptions

* Images of cars will be taken from similar angles; cars will be placed at the centre of the frame.
* The folder names match the exact model and make of the car.
* Each class will have a balanced distribution of training images.

Task B – Implementation of VMMR system

Code and output for Task B are available in the appendix under title **Appendix A: Code and Output for VMMR system implementation**. A downloadable file of the MATLAB .mlx file and a zip file with real-world data is available for download. Links have been placed in the appendix.

The MATLAB file was written using MATLAB Version R2020a

Task C – Test VMMR system of car dataset

Code and output for Task C in available in the appendix under title **Appendix A: Code and Output for VMMR system implementation**

# Discussion of testing, implementation, and results

Analysis of car dataset

Before testing the dataset on the VMMR system a careful analysis of the dataset was done to gain a understanding of what kind of image detail and quality is available. It was established that there was very poor data diversity, many of the images had undergone data augmentation in the form of rotations, skews, brightness levels altered, images had been cropped, and there were occlusions placed. **Figure C**  shows a table showing this augmentation with examples.

The outcome of the analysis indicated that the overall quality of the dataset was poor. With so many duplicate images it risks damaging the integrity of the model, as partitions of training and testing sets could very likely hold the same image data, leading to an artificial accuracy of the model.

To improve the quality of the dataset no manual alterations were made, instead an algorithm was written to sift through the dataset and only selected the full-scale images of cars, this was done through setting a threshold value on the row size for the image matrix, effectively selecting images that were only above a certain size. All full-scale images were at the same resolution of 640x480, through this algorithm all augmented and duplicated data was effectively removed from the dataset for training and testing. **Figure D** shows pseudocode and explanation of this algorithm.

Total number of images in the dataset before data cleaning : **2101**

Total number of images in the dataset after data cleaning : **527**

**Figure C – Analysis of augmentation in dataset**

|  |  |
| --- | --- |
| Data Augmentation | Image |
| Baseline Image example – no augmentation |  |
| Resize and greyscale filter |  |
| Image cropped and greyscale filtered |  |
| Image cropped and resized |  |
| Brightness levels altered (different car from baseline image) |  |
| Black borders around image (different car from baseline image) |  |
| Occlusion (different car from baseline image) |  |

**Figure D – Pseudocode for data cleaning**

newDataset = oldDataset % To not tamper with base dataset

finalDataset = imageDatasetObject %define new imagedatastore object

newLabels = categoricalArray % Define new categorical data array

amountOfImages = count(oldDataset.images)

for i = 1:amountOfImages % Loop to check eatch file for size

newLables = oldLabels(i) %get the file label

newImage = oldImage(i) %get the image filename

readImage(newImage) %get the file image matrix

imageRowSize = rowSize(newImage) %check the size of the matrix

%if the matrix has more than 300 rows

%add the label and file name to new dataset

if imageRowSize > 300

finalLabels = concatinate(1D, finalLabels, oldLabels(i))

finalDataset = concatinate(1D, newImage, finalDataset)

end

% Flip the labels to match the files

newLabels = flip(newLabels)

% Add the labels to the new dataset

finalDataset.Labels = newLabels

end

The actual code for this algorithm is available in the appendix under the title **Cleaning of the Dataset.**

Pre-processing the cleaned dataset

Simple pre-processing was applied to the cleaned dataset to prepare the data to train and test the model, **Figure E** shows a table outlined the pre-processing stages with results after each pre-processing step.

**Figure E – Table of pre-processing steps with explanations**

|  |  |  |
| --- | --- | --- |
| Pre-processing Stage | Explanation | Image |
| Baseline Image | This is the baseline image depicting a red car from the front. |  |
| Turn image greyscale | The image has been transformed into greyscale, this was done by firstly checking if the image has 3 channels (if it is RGB) if it does it triggers an if statement to transform the image into greyscale |  |
| Crop image to only show front features | The image has been cropped to depict only the front features of the car including the headlights, grill, bumper and badge. |  |
| Resize image to [70x140] | The image has been resized to 70x140, this was done for all images to create a standard of image size for training the model |  |
| Apply Histogram equalisation | Hisogram equlisation was applied to the image to create consisiancy of contrast across all images in the dataset |  |
| Remove noise (median filter) | Noise was removed from all images, in this image there is little noise however it was important to keep all images uniform, with reduced noise HOG will be able to extract for accurate features. |  |

**Figure F – Random sample of unprocessed and processed data**

|  |  |
| --- | --- |
| Unprocessed data | Processed data |
|  |  |

# Experiments

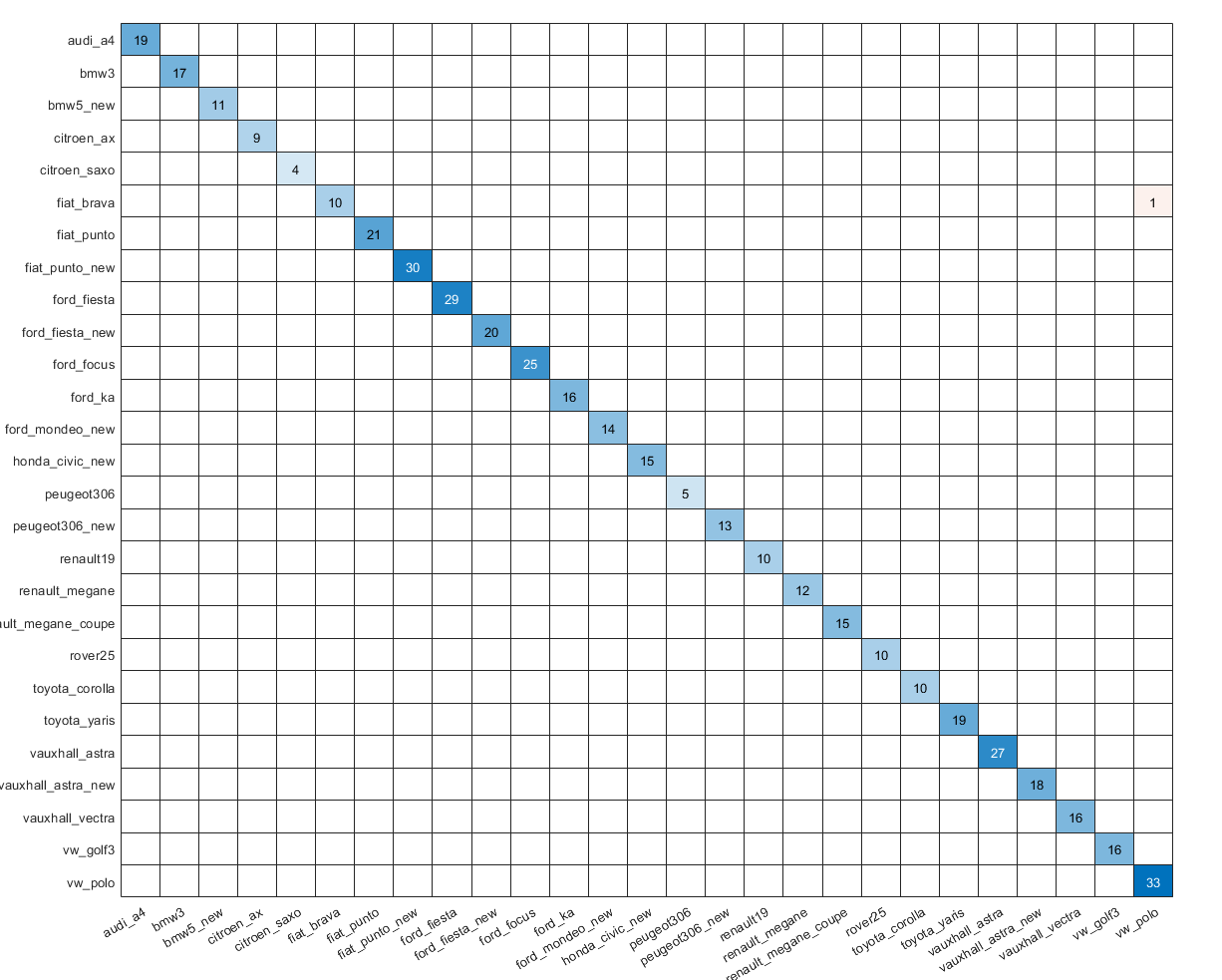
Several experiments were conducted on the VMMR with the car dataset, these were done to help establish the best parameters for the final model, different parameter setting and pre-processing steps were applied as well as different methods of cleaning the dataset.

## Bag of features model

Before any development was made on the VMMR and any alterations were done to the dataset a dummy model was made using a [*bagOfFeatures*](https://www.mathworks.com/help/vision/ref/bagoffeatures.html)SVM model. This model simply partitioned the dataset into training and testing, encoded the data into a bag of features, and trained a ECOC SVM classifier on the data. When testing the model on the test data it achieved an accuracy of 23% - this was also likely an artificial accuracy as the duplicate files still existed and were likely in both partitions of data, this result made it clear that further feature extraction, pre-processing and dataset cleaning was required to achieve a model that was viable. To achieve this model the following [documentation](https://www.mathworks.com/help/vision/ug/image-classification-with-bag-of-visual-words.html) was followed.

## Model with little data cleaning

The first successful ECOC SVM model built with HOG feature extraction was made with very little data cleaning in the car dataset, the only cleaning that was applied was the deletion of the full-scale images of the cars. When ECOC estimated error fluctuated between 1 and 3% clearly indicating that overfitting was occurring, however when analysing the structure and troubleshooting the architecture of the VMMR there was no indication of overfitting. **Figure G** shows a confusion matrix of predictions made on the test data by this model. It can be observed that the model accuracy that there is clearly an issue with the model, this is likely due to the duplicate images in the dataset appearing in both the training and testing data, so the model is effectively getting trained on the data it is being tested on which is causing it to achieve a very high level of accuracy. To verify this issue an expert’s opinion was obtained through contacting the lab assistant who explained some troubleshooting steps which would indicate if the model is overfitting or not, the results from this indicated that the issue was most likely occurring due to poor data diversity and distribution. 10-fold cross validation was also implemented for this model which meant that there was a higher chance of a duplicate image ending up being used for training and testing data. This model and all subsequent models utilised HOG feature selection as the method of feature extraction, this was decided due to the benefits for this use case that HOG can provide. HOG builds histograms of orients around edge it detects in an image, this is beneficial over just using a image matrices to train a model as it greatly condenses meaningful information, which directly reduces computational cost and also focuses on features on the objects rather than potential noise and the background of the image.



**Figure G – Confusion matrix and accuracy of SVM model with no data cleaning**



## Testing the model on Real World data

To get a reliable result for the quality and accuracy of this model, a new dataset of real-world testing data was created. This dataset included pre-cropped images of 27 cars, one for each class. **Figure H** shows an example of a real-world testing sample next to an image in the same class within the training dataset. This data was collected from the car sales website [AutoTrader](https://www.autotrader.co.uk/) . Each image was manually cropped in Photoshop to only show the specific features required, the background was removed, and the numberplate was blanked. The images had to be cropped manually as each car was in a slightly different position as the photographs had been taken from slightly different angles – this differs from the training dataset where the cars are roughly in the same location relative to one another.

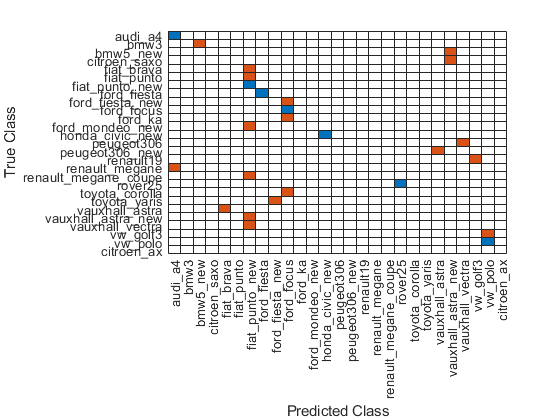
**Figure H – Real world test data compared with test data in the dataset**

|  |  |
| --- | --- |
| Real world test data – VW Polo (manually cropped) | Baseline car test data – VW Polo (cropped image in dataset) |
|  |  |

**Figure I** shows the confusion matrix of predictions and accuracy on real-world data from the non-data-cleaned SVM model. It can be observed from the accuracy and confusion matrix that the model does not perform very well on the new real-world testing images, the accuracy score for this test is far more reliable than the accuracy score on the testing data, indicating that the model is not viable.

**Figure I – Test accuracy and confusion matrix of non-cleaned SVM model on real world testing data**





Model with data cleaning

The model being presented as the final artifact of this project is the model with data cleaning, details regarding the data cleaning and pre-processing for this model are presented above in the **Analysis of car dataset** section.

For this model HOG feature selection was used to extract the features for training and testing, the optimal cell size for this was chosen to be [6 6] this was determined through testing various cell sizes. **Figure J** shows a table displaying HOG feature extraction with different cell sizes. The model was built with all these different cell sizes and it was established that the ‘sweet spot’ was a cell size of **[8 8]**, any higher cell count would result in slightly lower accuracy and any lower would not improve results but increase computational costs drastically, this can be observed in **Figure J**: cell size **[4 4]** does not find the features on the car any better than cell size **[6 6].** The test with cell size **[2 2]** took 3 hours to compile at constant 90%+ CPU usage. Models were built with cell size samples **[4 4]**, **[6 6]** and **[8 8]** to further verify the choice, the best result was with **[8 8]** at a training accuracy of 80% and real world accuracy 34%, **[6 6]** scored a testing accuracy of 78% and real world accuracy of 31% **[4 4]** scored a training accuracy of 67%, and real world accuracy of 27%, a model with cell size **[2 2]** was not run, as it was clear that further decreasing the cell size would not produce a more accurate model, and due to the expected run time and costly complication it was not worth testing. All of these models were ran with 10 fold cross validation.

**Figure J – Tests of different cell sizes on HOG feature selection**

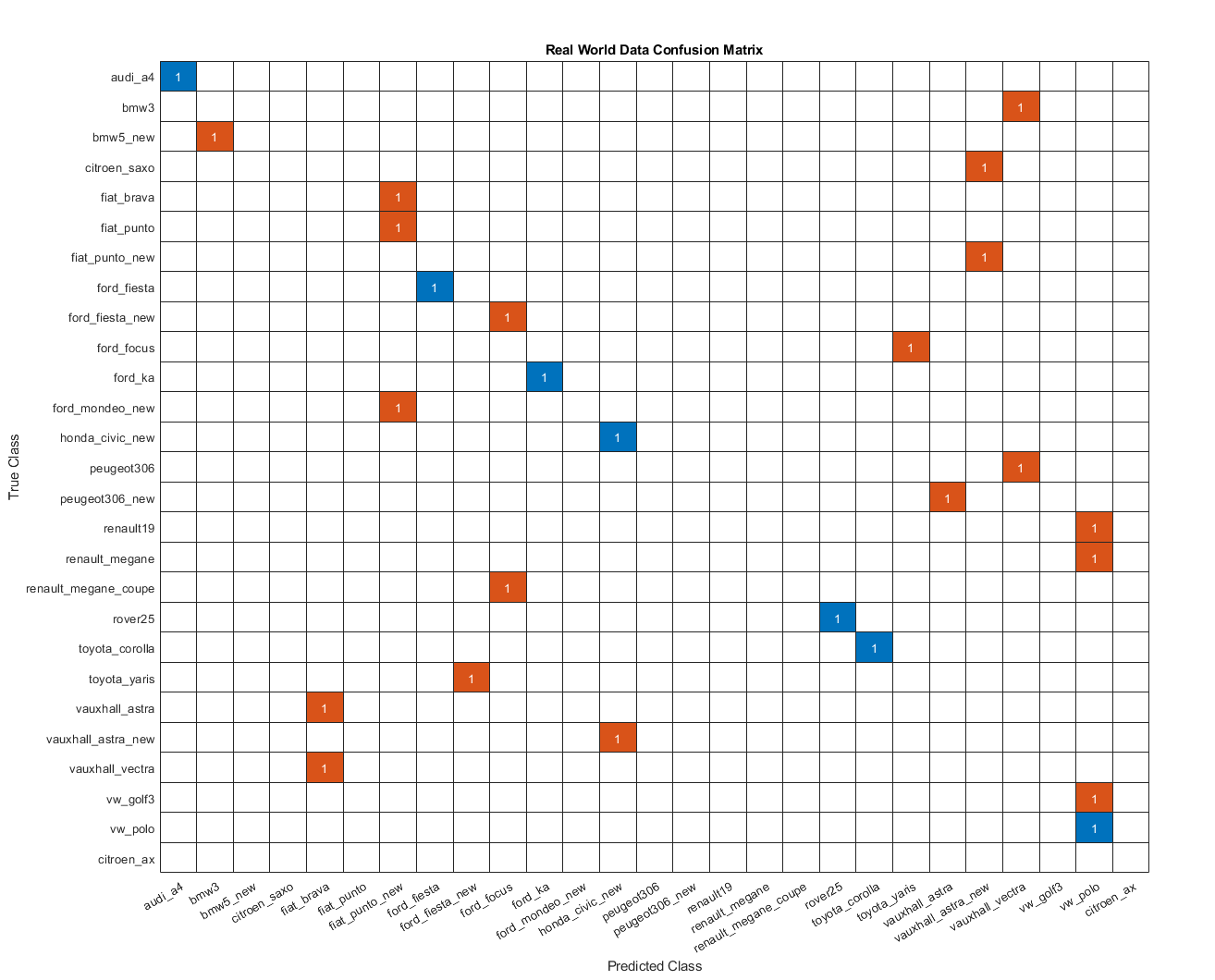
|  |  |  |
| --- | --- | --- |
| Cell size | Image and accuracy | Description |
| [2 2] |  | Far too many HOG features, large details such as the headlights are not being detected, very large computational cost in extracting features and compiling the model. |
| [4 4] |  | Too many HOG features, details are being picked up, large computational cost |
| [6 6] |  | Good amount of HOG features, details are being detected, medium computational cost |
| [8 8] |  | Good amount of HOG features, details are being detected, low computational cost |

## Analysis of best model accuracy

**Figure K** shows a confusion matrix of the testing scores achieved by the data-cleaned HOG feature SVM model on the testing data. It can be observed that the model that it performed relatively well on the testing data. Scores from this model fluctuated between 75-80% in different tests. However, when testing this model on the real-world dataset it only scored 33% accuracy, as seen in **Figure L**. This difference of accuracies may be a result of many different reasons. The images in the real-world testing dataset are captured at slightly different angles of the training images – the training image dataset is relatively uniform the full car images are taken from similar angles from one another, the images are of similar resolution and the cars are centred quite well – when finding the images for the real-world dataset all these variables were different in every case, the angle that the image is taken from is slightly different for each image, although the model was trained on images of the same vehicle make and models the perspective of the image is slightly different, which constitutes a slightly different HOG feature array. Another possibility is that there are still duplicate images in the training dataset, a possible way to mitigate this occurrence is to analyse the matrix of each image that is loaded in and remove the duplicate – however if any form of pre-processing such as a brightness increase has been applied to the image, the matrix will be slightly different, another alternative is to manually analyse each class folder and remove the duplicates by hand.

**Figure K – Confusion matrix of the data-cleaned model on the testing data**



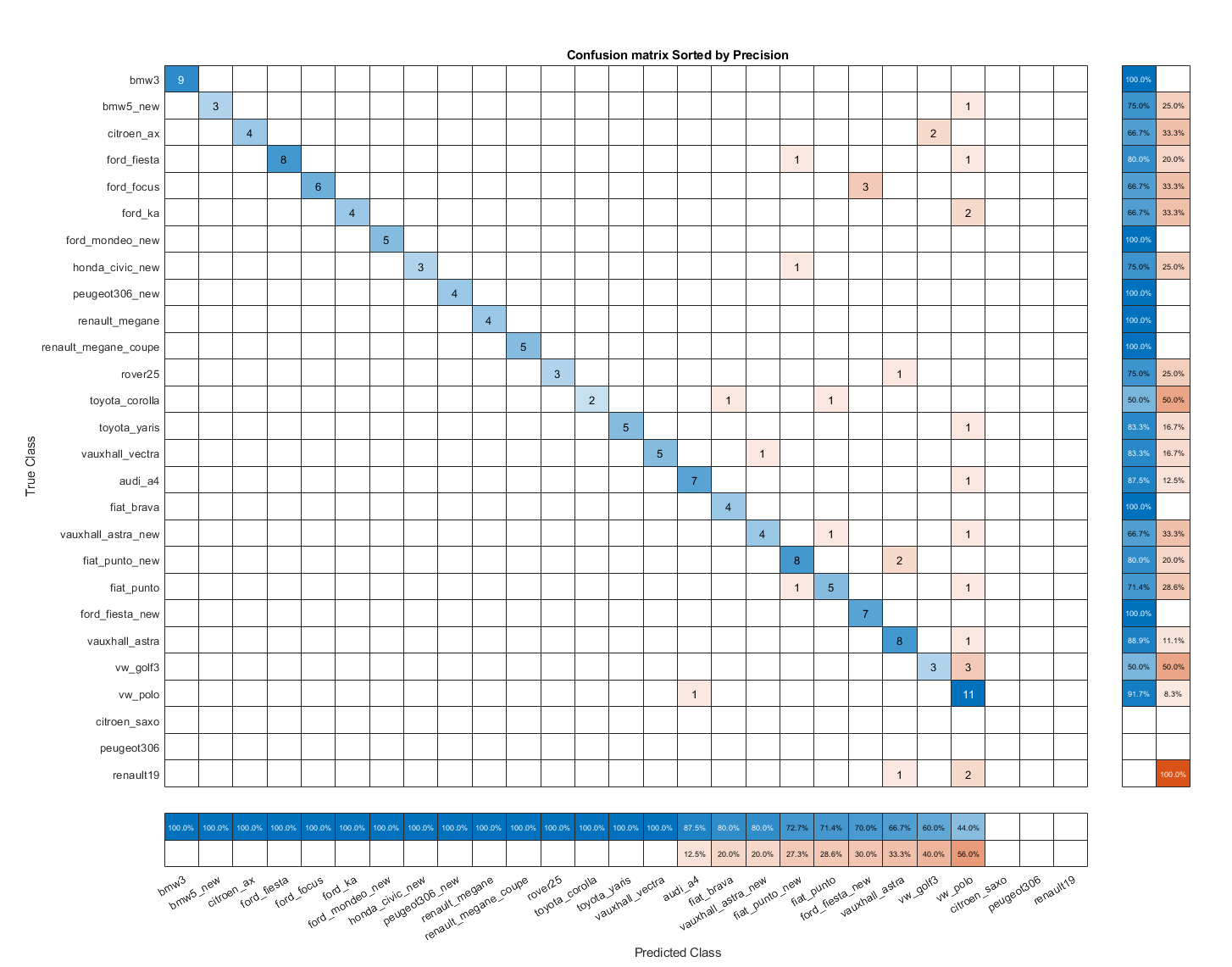
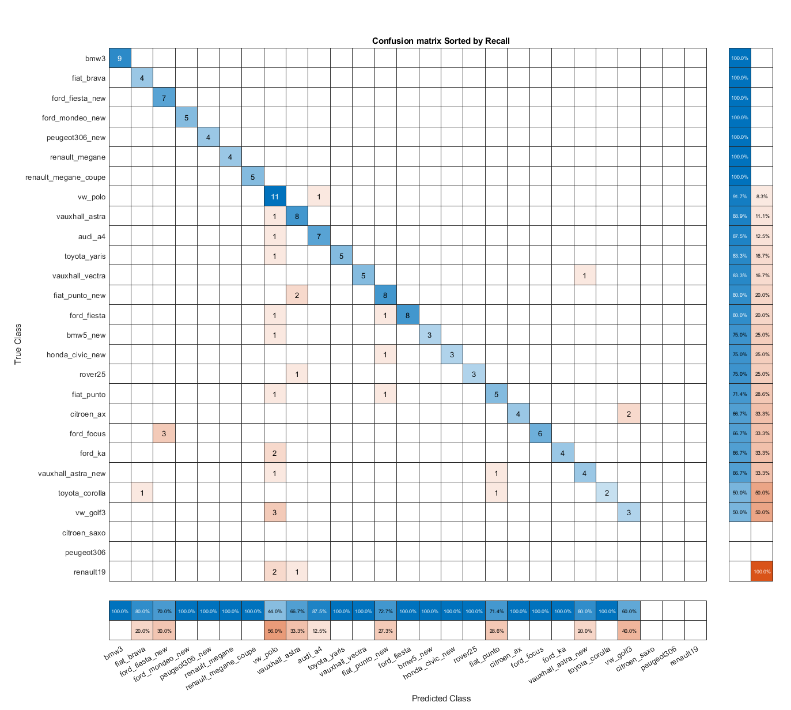


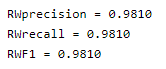
**Figure L – Confusion matrix of the data-cleaned model on the real-world dataset**

## Precision and recall of data cleaned model

**Figure M** shows confusion matrices of precision and recall of the model on the testing data along with their overall scores and F1 score, the scores were calculated through a function to compute each value through determining the true positives, false positive and false negatives. It can be observed from these metrics that the accuracy was very high and consistent for all classes apart from the citroen\_saxo, peugot306 and renault19 – scores are ranked in order in both confusion matrices.

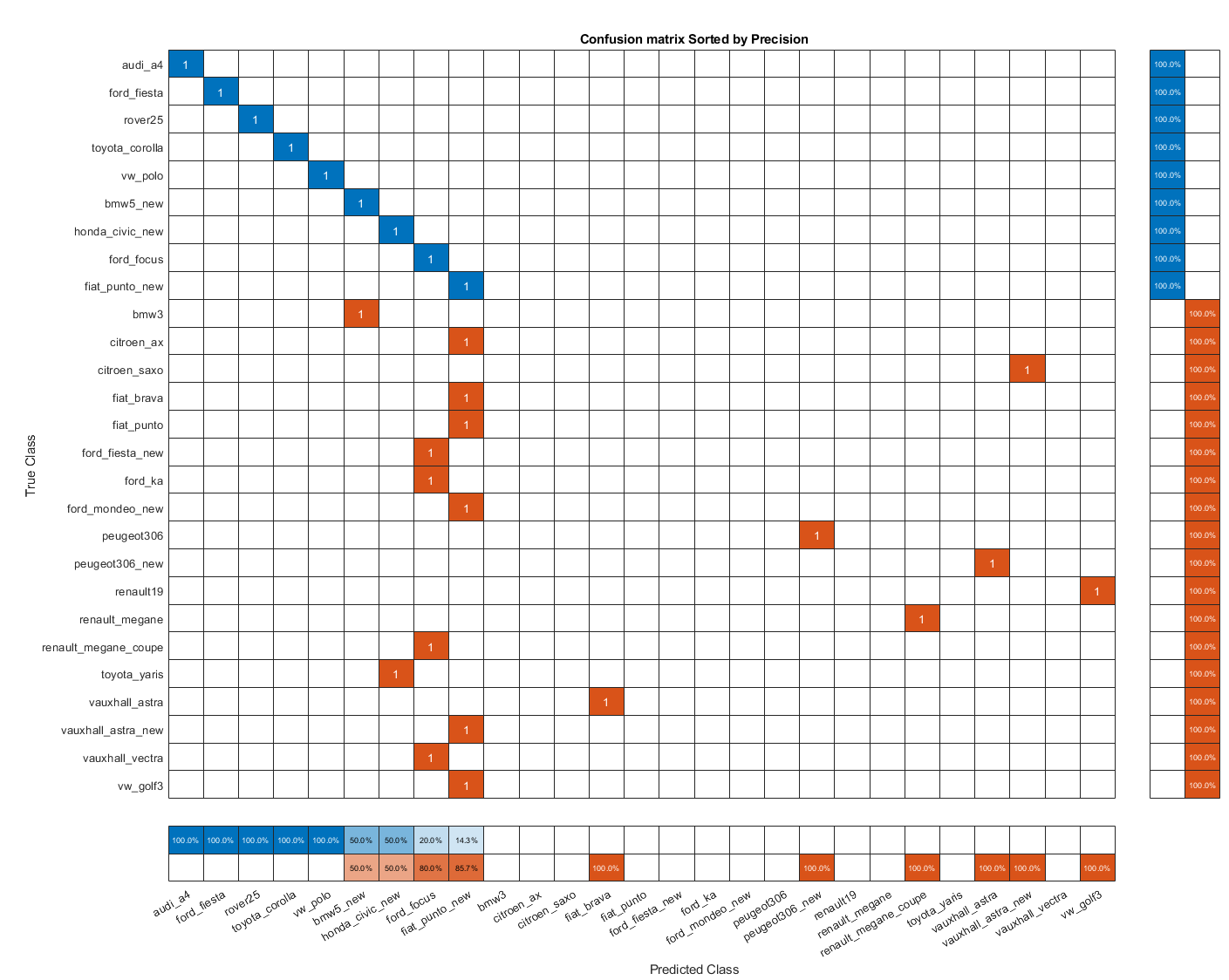
**Figure M** **– Precision, recall and F1 scores for the testing data**

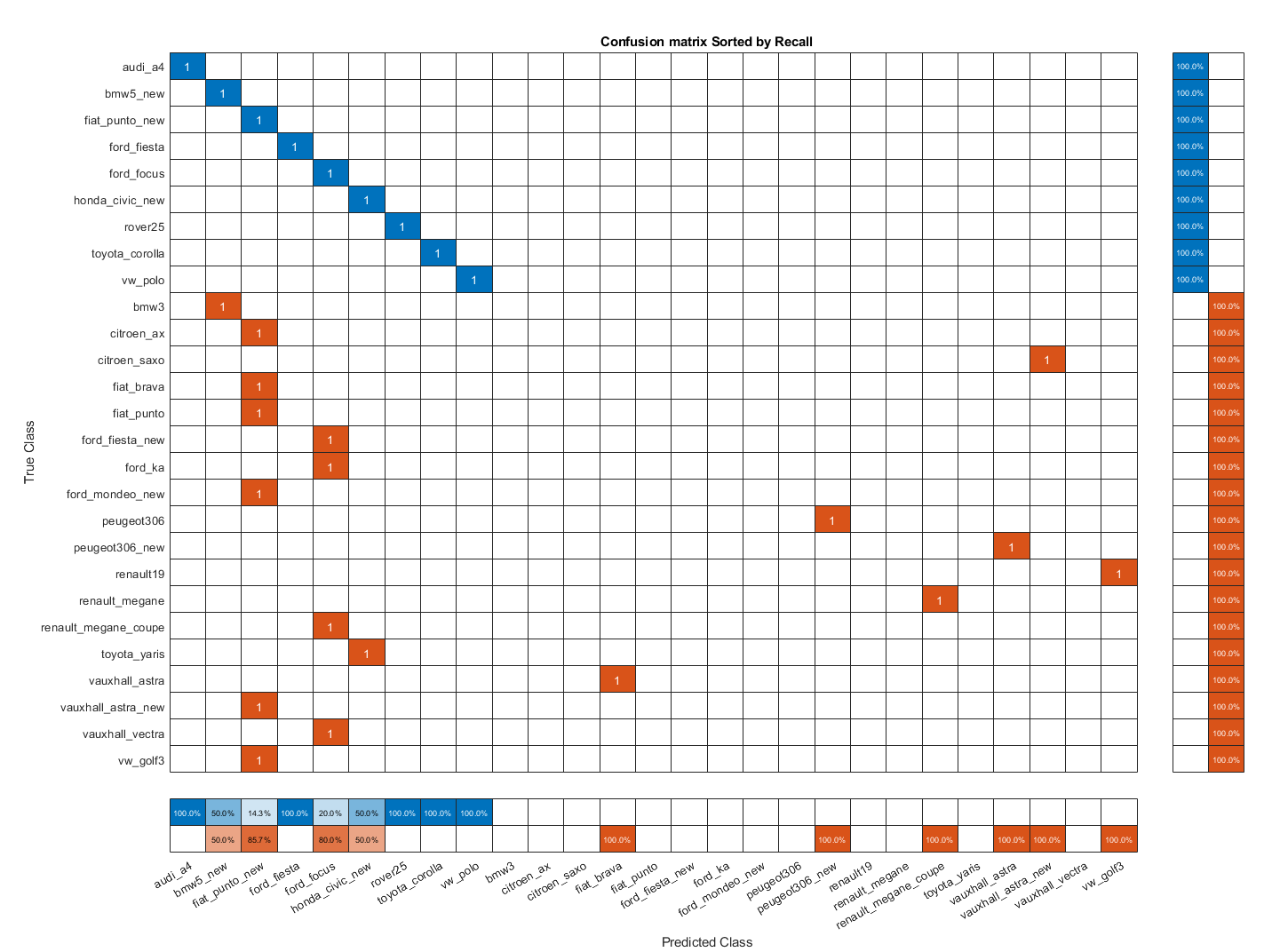




**Figure N** shows the recall and precision for the real-world data, it can be observed that on the real-world data the model performed much worse, recall and precision were also much lower due to many of the classes not being selected for classification.

**Figure N – Recall, Precision and F1 scores for the real-world data**







## Conclusions of the VMMR

The final iteration of the model achieved a training accuracy fluctuating between 75-80% which indicates that it may be a viable model for VMMR, however due to the much lower accuracy it achieved on the real-world testing data (30%) one conclusion that can be made is that images being classified on the model need to be captured at a similar angle to the images used to train the model. An alternative solution to this is to train the model with more diverse data – the total amount of images used for training was 369 images (roughly 13 images per class), these images were also taken from a similar angle, so features were in relatively similar positions, leading to a poor diversity of data.

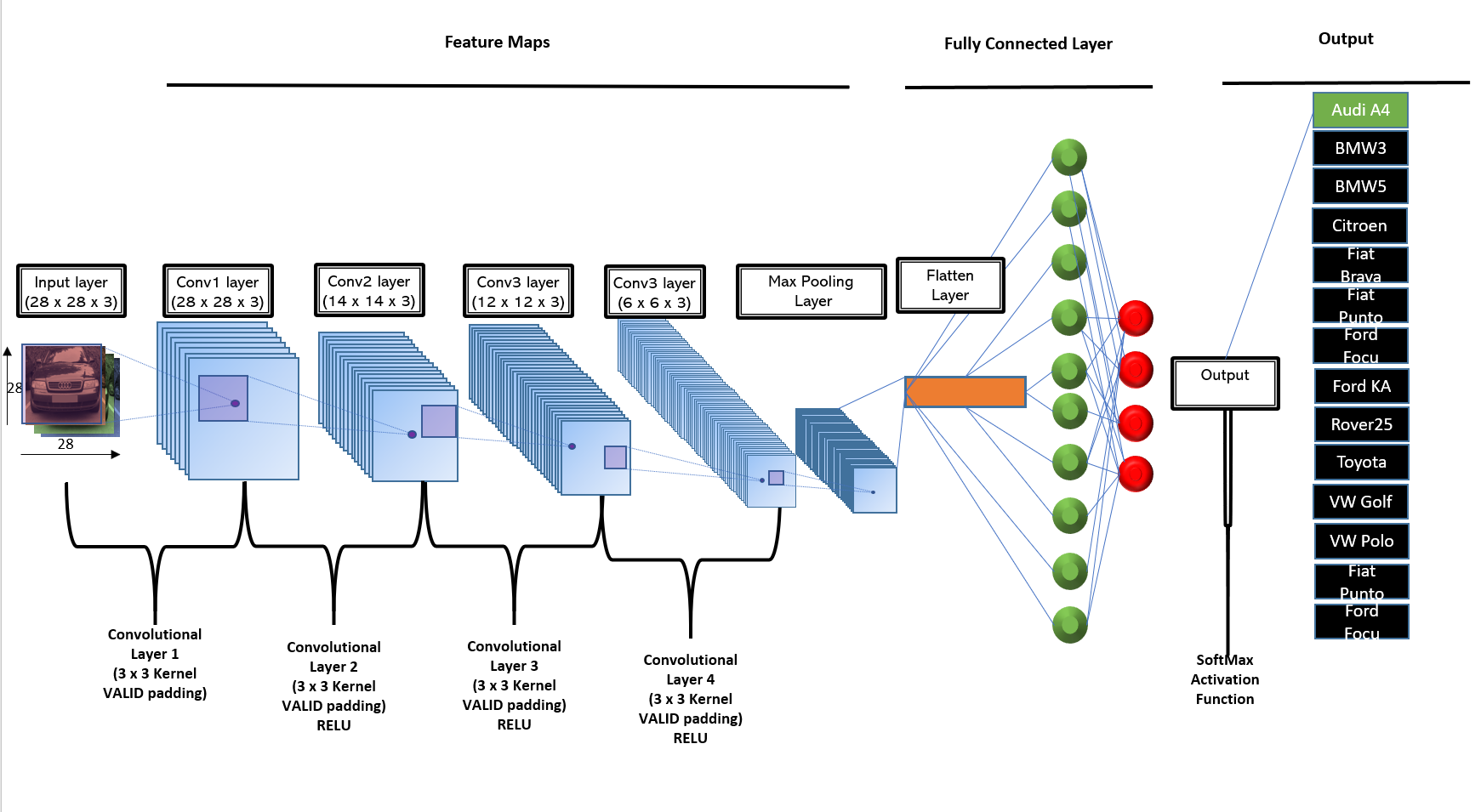
If the VMMR system were to be used in its current state it would require great normalisation of images that are being captured and classified, an ideal scenario for this is if there was a high-resolution camera mounted in a position with a motion capture trigger that would take photos of slow-moving cars from the front, so as they would match the images that the model was trained with. A better solution would be to train the model with more diverse data; therefore, it would be able to classify real-world examples from different angles. If more time was allocated to this project the next iteration would include diversifying the dataset as well as integrating functionality to detect duplicate images and remove them. A method of diversifying the dataset could be to collect more images from the internet, however as each image is taken by a different capturing device image resolution and position of the vehicle will be different in most cases. Another method could be to manually take photos of these different cars however as the make/models are quite old it could be hard to locate many samples. Data augmentation on this model is not incredibly useful, as it uses feature extraction to classify vehicles, slight augmentations to contrast, colour and brightness will not necessarily help in producing a higher quality model as the features will remain the same throughout eat augment. Augmentation in the form of skews and rotations are not necessarily useful either and could harm the accuracy of the model as the VMMR will receive input from a stationary camera taking the view of the front end of a vehicle as input, there is no need to skew or rotate the images as the model will very unlikely be used to classify vehicles with a positional relevance to the augmentation.

An additional iteration on top of this could be to incorporate more classes, the car make/models currently that the model can classify are not very common in most places. To choose which classes to add research would be done to evaluate which make/model combinations are most common in the area where the VMMR will be implemented. It may be advantageous to remove some of the classes of cars that are very uncommon today as it may result in lower overall accuracy for the classes of common cars as each addition of a class adds a slight increase on the margin of potential error. Many countries are passing legislation to ban older cars that produce more carbon emissions, this needs to be taken in consideration as in some places the older vehicle classes may not be needed and are taking up computational resources. Another consideration is that some cars may have modifications, if this system were to be used in conjunction with a ANPR (automatic number plate recognition) as misclassification may flag the car up as a stolen numberplate.

A further experiment that could be conducted to potentially produce a higher quality model would be to train a model with the full-scale images of the cars after they undergo segmentation. HOG feature selection works incredibly well in classifying objects of the same category (humans wearing different clothes for example) as the algorithm focuses on the edges of the object. One issue that the current dataset potentially has is that the cars are all too similar in design, as all objects have the same features with just slightly different shapes and angles. This could be a reason why the first few models achieved such a low accuracy on the real-world data. Training the network with the whole image of the car adds a few more potential features that HOG could detect, for instance the curvature of the bonnet or shape of the wing-mirrors. However, this may also negatively affect the quality of the model as it increases the margin of error as there will be more variables at play.

Task D – Convolutional solution for a VMMR

**Figure O** **– Diagram of proposed CNN for VMMR**



**Figure O** shows the proposed CNN (convolutional neural network) architecture for the VMMR. The input for the CNN is a resized image of a car in RGB format, for the model the image has been resized to (28x28) this is to improve performance when training the model. The CNN has 4 convolutional layers of different granularity of feature extraction. The model has multiple convolutional layers to allow hierarchical decomposition of the inputted image, as the model progresses each subsequent convolutional layer will extract lower-level features.

Each convolutional layer has a kernel size of 3x3 – the kernel is essentially a filter to extract features from the image, the size of 3x3 was chosen to keep feature extraction detail high but also keep computational cost low.

After each convolutional layer there is a ReLU activation function – ReLU helps prevent exponential growth with computational cost by removing all input that is below 0. ReLU was the chosen activation function to occur after each convolutional layer to avoid running into the vanishing gradient problem.

Once the data has gone through each convolutional layer it is parsed to a pooling layer, in the pooling layer the data undergoes dimension reduction – essentially reducing the numbers of parameters that the model must learn, this also reduced the computational cost, the pooling layer essentially summarises the features present in the data.

Then the data is parsed to the next layer where it is flattened, this means that it is transformed into a 1D array.

Regularisation is then done to the data in the form of random dropout, which penalises weight matrices of specific nodes; this is performed to reduce the chances of overfitting.

Then the data is parsed to a SoftMax activation function. The activation function was chosen as SoftMax as this activation function can result in multinomial probability distribution (not shown in diagram). This activation function is used to make the decision on what class the input is predicted.

Finally, there is the output of the CNN which is the class the model predicted, due to the usage of SoftMax the model will give a score for each class.

Task E – Increasing data diversity.

## Data Augmentation

Data augmentation is a technique that enables rapid and autonomous production of new data by performing augmentations to existing data. Typical data augmentation techniques for image classification include adjustments of contrast, brightness, sharpness, rotations, posterisations, solarisations, equalisations, colour changes, skewing data, cropping data, inverting data and many more.

In the context of a CNN (convolutional neural network) VMMR model not many of these augmentation techniques are useful, and in fact using some of them could harm the quality of the model by producing unrealistic images, for instance there is no use rotating a car 180 degrees as cars are generally always wheel side down.

One data augmentation method that could be used on this dataset would be to adjust the brightness and contrast of the images, this could be performed by setting a threshold maximum and minimum contrast and brightness settings then randomly picking a value of each to apply to images. This would help the model in classifying images that were taken of vehicles in different environments where there are varying levels of brightness, the contrast augmentations would have the same outcome, but might also offer the additional benefit of training the model to classify images taken with a different camera, as different cameras will produce images with slightly different contrast.

Another data augmentation technique that could be applied to the dataset before training a CNN model with it could be to randomly apply noise to images, this would train the network to be able to identify images that are noisy. Many different types of noise could be added to images, some examples are Gaussian noise, Poisson noise, and salt and pepper noise. Training the model on this augmented data could help the model with classifications of noisy data.

Another method of data augmentation that could be applied to the dataset is to resize and crop images, this could be done by setting maximum and minimum values for the resizes and crops. With cropped images the network could learn to classify partially visible car make and models and with resizes the model could learn to classify images of varying resolution; this would help make the CNN mode more useable with data from different sources and of different levels of quality.

## Transfer learning

Transfer learning is a technique in machine learning where an already built or partially built model can be used as a starting point for a new model, this method can help improve the quality of a model or accelerate training. In the context of a CNN built for the task of VMMR the pretrained model [AlexNet](https://en.wikipedia.org/wiki/AlexNet)) can be used as a baseline model for the VMMR task. AlexNet is a CNN network built with 15 million high-resolution images and can classify 1000 object categories.

To use AlexNet in conjunction with the VMMR CNN, the final 3 layers of the AlexNet CNN need to be fine-tuned for the new task of VMMR, to do this they must be replaced with a fully connected layer, a SoftMax layer and a classification output layer. AlexNet takes input images of (227x227x3) so the Images would have to undergo resizing to match this, among the resizing to slightly increase the diversity of data, augmentation could be done, viable methods of augmentation were discussed in the section above. The model would then be fit with the car dataset as validation data.

Appendix

# Appendix A: Code and Output for VMMR system implementation (1646 words)

## Code for VMMR System

MATLAB file (.mlx): <https://drive.google.com/file/d/1tZGVrUYvoheI5F03d0BUanAXXVtQyE3I/view?usp=sharing>

Real world Dataset (.zip) : <https://drive.google.com/file/d/1W0u3PoOESmPrvzawhQ0cYmyDPv2sM1I-/view?usp=sharing>

Vechile Make and Model Recognition

#### By Leon Davies

*This file was ran as it is with no modifications on the initial dataset downloaded from the module coursework page*

## Setup of file

filePath = 'og'; %filepath for dataset of car images

carDataset = imageDatastore(filePath,'IncludeSubfolders',true,'LabelSource','foldernames','FileExtensions', {'.jpg'}, 'ReadFcn', @processImages); %reads images in folders, takes folder names as labels

%carDataset = shuffle(carDataset); %Shuffles data in car dataset

% carDataset = cleanDataset(carDataset);

%imageOne = read(carDataset) % Show matrix of first image in dataset

totalImages = numel(carDataset.Files)

totalImages = 1649

%Testing of black border removal

disp("Unprocessed Image")

Unprocessed Image

borderImage = carDataset.Files{50}; %Car image with black border

imshow(borderImage) % Shows car image with black border



## Cleaning of the Dataset (removing already processed images)

%This section removes all pre-processed images from the dataset, based on

%if they are below a certain size

cleanImages = carDataset; % To not tamper with base dataset

cleanedDataset=imageDatastore({}); %define new imagedatastore object

labels = ({}); % Define new categorical data array

fileCount = numel(cleanImages.Files); %amount of files in dataset

for i = 1:fileCount % Loop to check eatch file for size

cleanLabel = cleanImages.Labels(i); %get the file label

imageInput = cleanImages.Files{i};%get the file name

readInput = imread(imageInput);%get the file image matrix

[rowCount, columnCount, channelCount] = size(readInput); %check the size of the matrix

%Delete images that have already been proccessed -based on if they have been cropped

if rowCount > 300 % if the matrix has more than 300 rows

%add the label and file name to new dataset

labels = cat(1, labels, cleanImages.Labels(i));

cleanedDataset = imageDatastore(cat(1,cleanImages.Files{i}, cleanedDataset.Files));

end

end

% Flip the labels to match the files

flippedLabels = flip(labels);

% Add the labels to the new dataset

cleanedDataset.Labels = flippedLabels;

## Test if pre-processed elimination was successful

%This displays 4 random images + labels from the new dataset

% The purpose of this is to test if the elimination of pre-processed

% data was sucessful

newFileCount = numel(cleanedDataset.Files);

randomDataGrab = randi([1 newFileCount], 1, 4); %random number beetwen 1-total files

figure;

subplot(2,2,1)

imshow(cleanedDataset.Files{randomDataGrab(1)})

title(cleanedDataset.Labels(randomDataGrab(1)))

subplot(2,2,2)

imshow(cleanedDataset.Files{randomDataGrab(2)})

title(cleanedDataset.Labels(randomDataGrab(2)))

subplot(2,2,3)

imshow(cleanedDataset.Files{randomDataGrab(3)})

title(cleanedDataset.Labels(randomDataGrab(3)))

subplot(2,2,4)

imshow(cleanedDataset.Files{randomDataGrab(4)})

title(cleanedDataset.Labels(randomDataGrab(4)))



## Splitting of data

%rng("default") %uncomment for reproduceability

disp("Splitting data")

Splitting data

trainSplit = 0.7

trainSplit = 0.7000

testSplit = 0.3

testSplit = 0.3000

[trainData,testData] = splitEachLabel(cleanedDataset,trainSplit,testSplit, 'randomized');

%View amount of images per class

disp("Amount of each class in Training set")

Amount of each class in Training set

countEachLabel(trainData) %training set

ans = 25×2 table

|  | **Label** | **Count** |
| --- | --- | --- |
| **1** | audi\_a4 | 20 |
| **2** | bmw3 | 21 |
| **3** | bmw5\_new | 9 |
| **4** | citroen\_ax | 15 |
| **5** | fiat\_brava | 8 |
| **6** | fiat\_punto | 17 |
| **7** | fiat\_punto\_new | 25 |
| **8** | ford\_fiesta | 23 |
| **9** | ford\_fiesta\_new | 16 |
| **10** | ford\_focus | 20 |
| **11** | ford\_ka | 13 |
| **12** | ford\_mondeo\_new | 11 |
| **13** | honda\_civic\_new | 11 |
| **14** | peugeot306\_new | 10 |
| **15** | renault19 | 7 |
| **16** | renault\_megane | 10 |
| **17** | renault\_megane\_coupe | 13 |
| **18** | rover25 | 8 |
| **19** | toyota\_corolla | 8 |
| **20** | toyota\_yaris | 14 |
| **21** | vauxhall\_astra | 22 |
| **22** | vauxhall\_astra\_new | 15 |
| **23** | vauxhall\_vectra | 13 |
| **24** | vw\_golf3 | 13 |
| **25** | vw\_polo | 27 |

disp("Amount of each class in Testing set")

Amount of each class in Testing set

countEachLabel(testData) %testing set

ans = 25×2 table

|  | **Label** | **Count** |
| --- | --- | --- |
| **1** | audi\_a4 | 8 |
| **2** | bmw3 | 9 |
| **3** | bmw5\_new | 4 |
| **4** | citroen\_ax | 6 |
| **5** | fiat\_brava | 4 |
| **6** | fiat\_punto | 7 |
| **7** | fiat\_punto\_new | 10 |
| **8** | ford\_fiesta | 10 |
| **9** | ford\_fiesta\_new | 7 |
| **10** | ford\_focus | 9 |
| **11** | ford\_ka | 6 |
| **12** | ford\_mondeo\_new | 5 |
| **13** | honda\_civic\_new | 4 |
| **14** | peugeot306\_new | 4 |
| **15** | renault19 | 3 |
| **16** | renault\_megane | 4 |
| **17** | renault\_megane\_coupe | 5 |
| **18** | rover25 | 4 |
| **19** | toyota\_corolla | 4 |
| **20** | toyota\_yaris | 6 |
| **21** | vauxhall\_astra | 9 |
| **22** | vauxhall\_astra\_new | 6 |
| **23** | vauxhall\_vectra | 6 |
| **24** | vw\_golf3 | 6 |
| **25** | vw\_polo | 12 |

## Data processing

% Show matrix of first image in dataset before resize

resizedImageOne = read(trainData);

%Values for iterators

trainCount = numel(trainData.Files);

testCount = numel(testData.Files);

%Define empty cell arrays

processedTrainData = ({});

processedTestData = ({});

% Calls preprocessing function into through readall

% Removes black borders from images through cropping

% Resizes images so same size (70x140)

% Performs histogram equalisation

% Removes noise with a median filter

% true parameter chooses if images are to be cropped

for i = 1:trainCount

processedTrainData {i} = processImages(trainData.Files{i}, true);

end

for i = 1:testCount

processedTestData {i} = processImages(testData.Files{i}, true);

end



%This can be used to verify pre-processing was sucessful

% Show matrix of first image in dataset after pre-processing

resizedImageOne = read(trainData);

%Transpose data

processedTrainData = processedTrainData';

processedTestData = processedTestData';

## Samples of training/testing data

%Random sample of training data

disp("Random sample of training data")

Random sample of training data

showSamples(processedTrainData);

Total amount of images in dataset: 369  
 "Displaying index location: 244"  
 "Displaying index location: 9"  
 "Displaying index location: 10"  
 "Displaying index location: 186"



%Random sample of test data

disp("Random sample of testing data")

Random sample of testing data

showSamples(processedTestData);

Total amount of images in dataset: 158  
 "Displaying index location: 123"  
 "Displaying index location: 130"  
 "Displaying index location: 9"  
 "Displaying index location: 25"



## HOG feature extraction

% Extract HOG features from Training set

trainingCount = numel(processedTrainData)

trainingCount = 369

% Loops for each image in the training dataset

for i = 1:trainingCount

inputImage = processedTrainData{i};

%Extracts HOG features

[HOGFeatures(i, :), HogVis(i,:)] = extractHOGFeatures(inputImage,'CellSize', [8 8]);

end

%Shows a figure with an image with its HOG counterpart interlaced

figure;

imshow(processedTrainData{i}) %image

hold on

plot(HogVis(i)) %HOG plot



## Fitting model

% Stores training labels in variable

trainLabels = trainData.Labels;

%Fits model (X = transformed images, Y = labels of images)

mdl = fitcecoc(HOGFeatures, trainLabels);

%10 fold cross validation

Xvalidation = crossval(mdl);

Warning: One or more of the unique class values in GROUP is not present in one or more folds. For classification problems, either remove this class from the data or use N instead of GROUP to obtain nonstratified partitions. For regression problems with continuous response, use N.

%Estimates classification error

errorEstimate = kfoldLoss(Xvalidation)

errorEstimate = 0.2033

Evaluation of model on Testing data

testingCount = numel(testData.Files) %Amount of images in test data

testingCount = 158

testingLabels = testData.Labels; % Corresponding labels to test data images

% Extract HOG features from testing data

for i = 1:testingCount

inputImage = processedTestData{i};

testingFeatures(i,:) = extractHOGFeatures(inputImage, 'CellSize' ,[8 8]);

end

## Visualisation of Testing data results

% Calculates the accuracy of the model

[testPredictions, testScore, testCost] = predict(mdl, testingFeatures); % Predicts the values of the test data

accuracy = sum(testData.Labels == testPredictions)/size(testData.Labels,1); %calculation to work out the accuracy

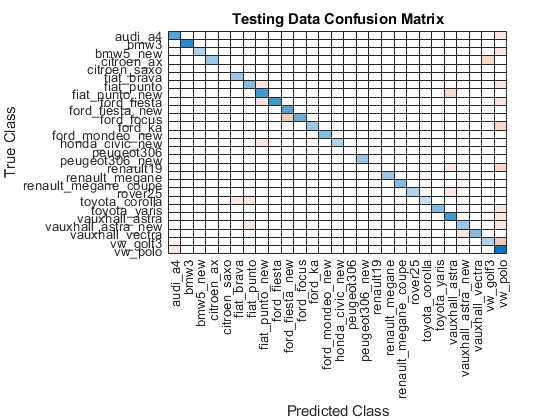
disp("Model accuracy: " + accuracy)

Model accuracy: 0.8038

% Plot confusion matrix of ground-truth testing labels against predictions

figure;

testingCM = confusionchart(testingLabels, testPredictions, 'Title', "Testing Data Confusion Matrix")

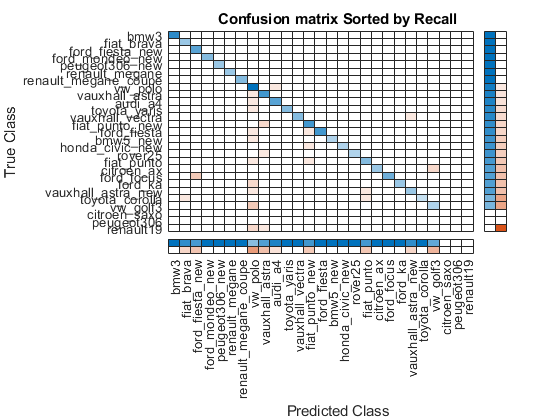


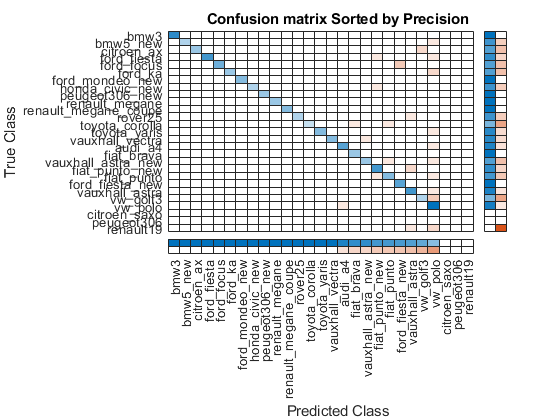
testingCM =

ConfusionMatrixChart (Testing Data Confusion Matrix) with properties:  
  
 NormalizedValues: [27×27 double]  
 ClassLabels: [27×1 categorical]  
  
 Show all properties

%Confusion matrices of recall and precision

[testRecall, testPrecision] = confusionMatrixMetrics(testingLabels, testPredictions);





% Calculates precision, recall, and F1 core for Realworld Data

[RWprecision, RWrecall, RWF1] = calculateScores(testingLabels, testPredictions)

RWprecision = 0.9810

RWrecall = 0.9810

RWF1 = 0.9810

## Evaluation of data on Real-world data

%Real world testing

realWorldPath = 'REALWORLD\_TESTING'; %filepath for dataset of car images

realWorld = imageDatastore(realWorldPath,'IncludeSubfolders',true,'LabelSource','foldernames','FileExtensions', {'.jpg'}); %reads images in folders, takes folder names as labels

realWorld = shuffle(realWorld);

RWdataCount = numel(realWorld.Files);

RWLabels = realWorld.Labels;

processedRWData = ({});

for i = 1:RWdataCount

processedRWData {i} = processImages(realWorld.Files{i}, false);

end



% Extract HOG features from real world data

for i = 1:RWdataCount

inputImage = processedRWData{i};

RWFeatures(i,:) = extractHOGFeatures(inputImage, 'CellSize' ,[8 8]);

end

## Visualisation of Real-world testing results

% Real world results

realWorldLabels = realWorld.Labels;

% Calculates the accuracy of the real world data

[realWorldPredictions, RWScore, RWCost] = predict(mdl, RWFeatures); % Predicts the values of the validation data

accuracy = sum(realWorldLabels == realWorldPredictions)/size(realWorldLabels,1); %calculation to work out the accuracy

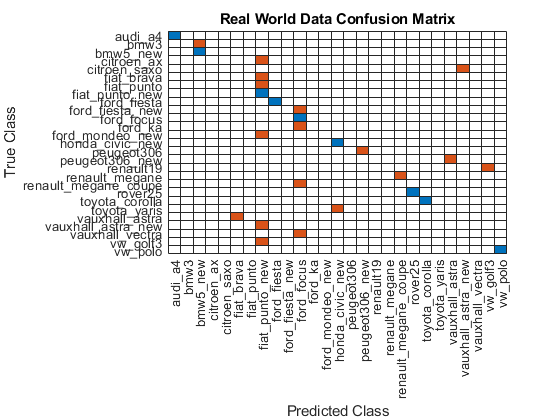
disp("Real world Testing accuracy: " + accuracy)

Real world Testing accuracy: 0.33333

% Plot confusion matrix of ground-truth real world labels against predictions

figure;

realWordCM = confusionchart(realWorldLabels, realWorldPredictions, 'Title', "Real World Data Confusion Matrix")

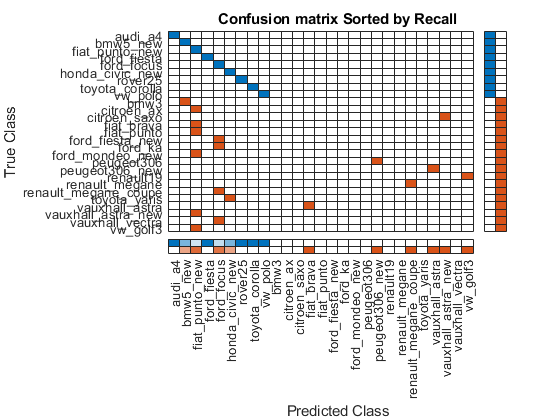


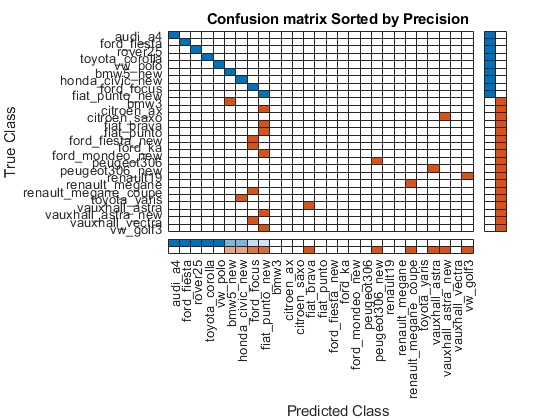
realWordCM =

ConfusionMatrixChart (Real World Data Confusion Matrix) with properties:  
  
 NormalizedValues: [27×27 double]  
 ClassLabels: [27×1 categorical]  
  
 Show all properties

% Confusion matracies of recall and precision

[testRecall, testPrecision] = confusionMatrixMetrics(realWorldLabels, realWorldPredictions);





% Calculates precision, recall, and F1 core for Realworld Data

[RWprecision, RWrecall, RWF1] = calculateScores(realWorldLabels, realWorldPredictions)

RWprecision = 0.5556

RWrecall = 0.5556

RWF1 = 0.5556

## Function to show 10 random samples of data

function [examples] = showSamples(dataset) %function that displays 4 random images and displays them in a 2x2 figure

totalImages = numel(dataset); %counts total number of files

disp("Total amount of images in dataset: " + totalImages)

%rng("default"); %comment out for reproduceability

dataRand = randi([1 totalImages], 1, 4); %creates array with 4 random values bettwen 1 and total number of files

disp("Displaying index location: " + dataRand(:))

examples = figure; %displays image files

subplot(2,2,1);

imshow(dataset{dataRand(1)}); %shows image of index location dataRand(n)

subplot(2,2,2);

imshow(dataset{dataRand(2)});

subplot(2,2,3);

imshow(dataset{dataRand(3)});

subplot(2,2,4);

imshow(dataset{dataRand(4)});

end

## Pre-processing function

function [removedNoise] = processImages(imageInput, crop)

%Load in image

rawImage = imread(imageInput);

imshow(rawImage)

%Size of image

[rowCount, columnCount channelCount] = size(rawImage);

%checks if image has 3 channels (RGB)

% If it does, it will be convereted to greyscale

if channelCount == 3

rawImage = rgb2gray(rawImage);

end

if crop == true

%crop image to just show front

rect = [90.5 135.5 462 208];

rawImage = imcrop(rawImage, rect);

end

%Resize image

resizedImage = imresize(rawImage,[70 140]); %resizes cropped image to 70x140

%Histograph Equalisation

equalised = histeq(resizedImage);

%Remove noise

removedNoise = medfilt2(equalised);

end

% Builds recall and precicion confusion matricies from input labels

% Code adapted from https://www.mathworks.com/help/deeplearning/ref/confusionchart.html#d123e23203

function [recallGraph, precicionGraph] = confusionMatrixMetrics(trueLabels, predictLabels)

figure;

%Recall graph

recallGraph = confusionchart(trueLabels,predictLabels, ...

'ColumnSummary','column-normalized', ...

'RowSummary','row-normalized', 'Title', "Confusion matrix Sorted by Recall");

%Parameters to display recall

recallGraph.Normalization = 'row-normalized';

sortClasses(recallGraph,'descending-diagonal');

recallGraph.Normalization = 'absolute';

figure;

%Precicion graph

precicionGraph = confusionchart(trueLabels,predictLabels, ...

'ColumnSummary','column-normalized', ...

'RowSummary','row-normalized', 'Title', "Confusion matrix Sorted by Precision");

%Parameters to display precicison

precicionGraph.Normalization = 'column-normalized';

sortClasses(precicionGraph,'descending-diagonal');

precicionGraph.Normalization = 'absolute';

end

% Function to calculate precision, recall and F1 score

function [precision, recall, F1] = calculateScores(correctLabels,predictLabels)

checkRelevant = ismember(correctLabels, predictLabels); % Check

%tabulate(checkRelevant);

entryCount = numel(predictLabels); % Total number of predictions

IDcount = numel(correctLabels); % Total number of labels

TP = sum(checkRelevant(:) == 1); % Relevant

FP = IDcount - TP; % Non relevant

FN = entryCount - TP; % False negative

precision = TP/(TP + FP); %Precision

recall = TP/(TP + FN); %Recall

F1 = (2\*precision\*recall)/(precision+recall); %F1 score

end